PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-076086
(43)Date of publication of application : 14.03.2000

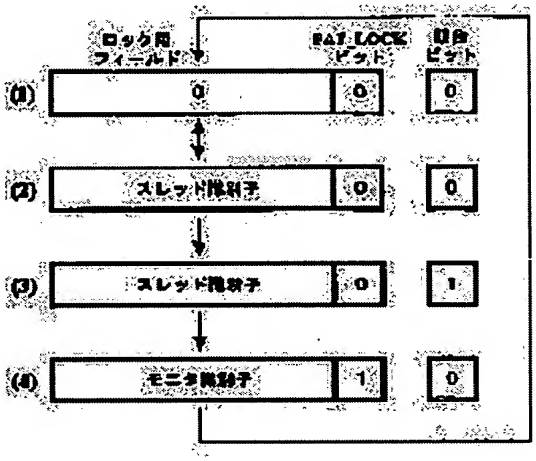
(51)Int.Cl. G06F 9/46

(21)Application number : 10-244497
(22)Date of filing : 31.08.1998

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>
(72)Inventor : ONODERA TAMIYA
KOCHIYA KIYOKUNI

(54) METHOD AND DEVICE FOR LOCK MANAGEMENT OF OBJECT AND METHOD AND DEVICE FOR UNLOCKING OBJECT

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a compound lock method not to decelerate the processing speed of a high-frequency path for locking, accessing and unlocking an object.
SOLUTION: In a state of making plural sleds existent, when managing lock to the object by storing a bit showing the kind of lock and the identifier of a sled, which acquires lock corresponding to the lock of a first kind, or the identifier of lock of a second kind in a storage area provided corresponding to the object, this method executes a step for judging the bit showing the kind of lock for a certain object held by the first sled shows the lock of the first kind or not when the second sled tries to acquire the lock of a certain object and a step for putting up a competition bit when the bit shows the lock of the first kind.



LEGAL STATUS

[Date of request for examination] 27.12.1999
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2000－76086
（P2000－76086A）
(43)公開日　平成12年３月14日(2000.3.14)

(51)Int.Cl.⁷	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ｈ　５Ｂ０９８

審査請求　未請求　請求項の数18　ＯＬ　（全 14 頁）

(21)出願番号　特願平10－244497

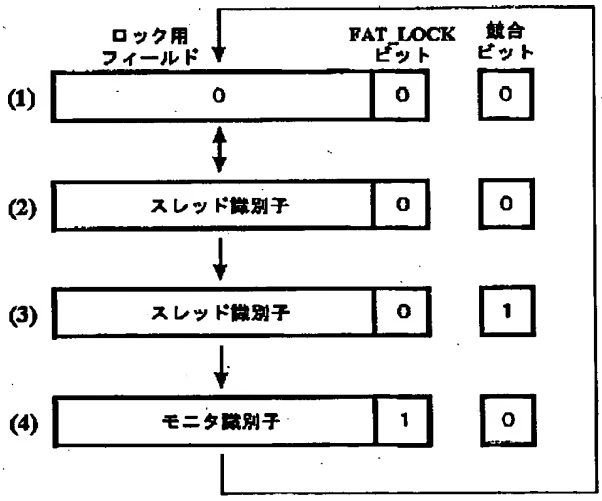
(22)出願日　平成10年８月31日(1998.8.31)

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）
(74)代理人　100086243
弁理士　坂口　博　（外１名）

最終頁に続く

(54)【発明の名称】　オブジェクトのロック管理方法及び装置、並びにオブジェクトのロック解除方法及び装置

(57)【要約】　（修正有）
【課題】オブジェクトにロックをし、アクセスし、アンロックするという高頻度パスの処理速度を低下させない、複合ロック方法。
【解決手段】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又は第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合に、第１のスレッドが保持しているあるオブジェクトへのロックを第２のスレッドが獲得しようとした場合、あるオブジェクトのロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、第１の種類のロックであることを示している場合には、競合ビットを立てるステップとを実行する。



1

【特許請求の範囲】

【請求項１】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又は第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する方法であって、

第１のスレッドが保持しているあるオブジェクトへのロックを第２のスレッドが獲得しようとした場合、前記あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、

前記第１の種類のロックであることを示している場合には、競合ビットを立てるステップと、

を含むロック管理方法。

【請求項２】前記第１のスレッドが前記あるオブジェクトへのロックを解除する際に、前記ロックの種類を示すビットが前記第１の種類のロックであることを示しているか判断するステップと、

前記あるオブジェクトのロックを保持しているスレッドが存在しないことを前記記憶領域に記憶するステップと、

前記ロックの種類を示すビットが前記第１の種類のロックであることを示している場合には、前記競合ビットが立っているか判断するステップと、

前記競合ビットが立っていないと判断された場合には、他の処理を実施せずにロック解除処理を終了するステップと、

をさらに含む請求項１記載のロック管理方法。

【請求項３】前記競合ビットが立っていると判断された場合には、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に前記第１のスレッドが移行するステップと、

待機しているスレッドへの通知操作を実行するステップと、

前記第１のスレッドが前記排他制御状態から脱出するステップと、

をさらに含む請求項２記載のロック管理方法。

【請求項４】前記第１の種類のロックとは、オブジェクトに対してロックを実施するスレッドの識別子を当該オブジェクトに対応して記憶することによりロック状態を管理するロック方式である、請求項１記載のロック管理方法。

【請求項５】前記第２の種類のロックとは、オブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である、請求項１記載のロック管理方法。

【請求項６】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する方法であって、

オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、

あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、

前記あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示している場合、前記第１の種類のロックを第１のスレッドが獲得できるか判断するステップと、

前記第１の種類のロックを第１のスレッドが獲得できる場合には、前記あるオブジェクトに対応して設けられた記憶領域に第２の種類のロックを示すビット及び前記第２の種類のロックの識別子を記憶するステップと、

を含み、前記第１のスレッドは前記あるオブジェクトに対して必要な処理を終了した後に前記排他的状態を脱出することを特徴とするロック管理方法。

【請求項７】前記第１の種類のロックを第１のスレッドが獲得できない場合には、前記機構の待機状態に移行するステップとをさらに含む請求項６記載のロック管理方法。

【請求項８】前記あるオブジェクトのロックの前記ロックの種類を示すビットが第１の種類のロックであることを示していない場合、前記第１のスレッドは前記第２の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施するステップと、

をさらに含む請求項６記載のロック管理方法。

【請求項９】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する方法であって、

オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、

あるオブジェクトの前記ロックの種類を示すビットが第２の種類のロックであることを示しているか判断する第１判断ステップと、

前記あるオブジェクトの前記ロックの種類を示すビット

3

が第２の種類のロックであることを示している場合、前記第１のスレッドは前記第２の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施するステップと、
を含むロック管理方法。
【請求項１０】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、前記ロックを解除する方法であって、
第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックであるか判断するステップと、
前記第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが存在しているか判断するステップと、
前記他のスレッドが存在していない場合、ロックを保持しているスレッドが存在しないことを前記記憶領域に記憶するステップとを含むロック解除方法。
【請求項１１】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又は第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する装置であって、
第１のスレッドが保持しているあるオブジェクトへのロックを第２のスレッドが獲得しようとした場合、前記あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示しているか判断する手段と、
前記第１の種類のロックであることを示している場合には、競合ビットを立てる手段と、
を含むロック管理装置。
【請求項１２】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する装置であって、
オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドを移行させる手段と、

4

あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示しているか判断する手段と、
前記あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示している場合、前記第１の種類のロックを第１のスレッドが獲得できるか判断する手段と、
前記第１の種類のロックを第１のスレッドが獲得できる場合には、前記あるオブジェクトに対応して設けられた記憶領域に第２の種類のロックを示すビット及び前記第２の種類のロックの識別子を記憶する手段と、
を含み、前記第１のスレッドは前記あるオブジェクトに対して必要な処理を終了した後に前記排他的状態を脱出することを特徴とするロック管理装置。
【請求項１３】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する装置であって、
オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドを移行させる手段と、
あるオブジェクトの前記ロックの種類を示すビットが第２の種類のロックであることを示しているか判断する手段と、
前記あるオブジェクトの前記ロックの種類を示すビットが第２の種類のロックであることを示している場合、前記第１のスレッドは前記第２の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施する手段と、
を含むロック管理装置。
【請求項１４】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、前記ロックを解除する装置であって、
第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックであるか判断する手段と、
前記第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが

5

存在しているか判断する手段と、
前記他のスレッドが存在していない場合、所定の条件を満たしているか判断する手段と、
前記所定の条件を満たしている場合に、ロックを保持しているスレッドが存在しないことを前記記憶領域に記憶する手段とを含むロック解除装置。
【請求項１５】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又は第２の種類のロックの識別子を記憶することによりオブジェクトへのロックをコンピュータに管理させるプログラムを記憶する記憶媒体であって、
前記プログラムは前記コンピュータに、
第１のスレッドが保持しているあるオブジェクトへのロックを第２のスレッドが獲得しようとした場合、前記あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、
前記第１の種類のロックであることを示している場合には、競合ビットを立てるステップと、
を実行させる、記憶媒体。
【請求項１６】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックをコンピュータに管理させるプログラムを格納する記憶媒体であって、
前記プログラムは前記コンピュータに、
オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、
あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、
前記あるオブジェクトの前記ロックの種類を示すビットが第１の種類のロックであることを示している場合、前記第１の種類のロックを第１のスレッドが獲得できるか判断するステップと、
前記第１の種類のロックを第１のスレッドが獲得できる場合には、前記あるオブジェクトに対応して設けられた記憶領域に第２の種類のロックを示すビット及び前記第２の種類のロックの識別子を記憶するステップと、
を実行させ、前記第１のスレッドは前記あるオブジェクトに対して必要な処理を終了した後に前記排他的状態を脱出することを特徴とする、記憶媒体。

6

【請求項１７】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックをコンピュータに管理させるプログラムを格納した記憶媒体であって、
前記プログラムは前記コンピュータに、
オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、
あるオブジェクトの前記ロックの種類を示すビットが第２の種類のロックであることを示しているか判断する第１判断ステップと、
前記あるオブジェクトの前記ロックの種類を示すビットが第２の種類のロックであることを示している場合、前記第１のスレッドは前記第２の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施するステップと、を実行させる、記憶媒体。
【請求項１８】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、前記ロックをコンピュータに解除させるプログラムを格納した記憶媒体であって、
前記プログラムは前記コンピュータに、
第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックであるか判断するステップと、
前記第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが存在しているか判断するステップと、
前記他のスレッドが存在していない場合、ロックを保持しているスレッドが存在しないことを前記記憶領域に記憶するステップとを実行させる、記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、複数のスレッドが存在し得る状態における、オブジェクトへのロックに関する。
【０００２】
【従来の技術】複数のスレッドが動作するプログラムで

7

オブジェクトへのアクセスを同期させるには、アクセスの前にオブジェクトをロック（lock）し、次にアクセスを行い、アクセスの後にアンロック（unlock）するようにプログラムのコードは構成される。このオブジェクトのロックの実装方法としては、スピンロック及びキューロックがよく知られている。また、最近ではそれらを組み合わせたもの（以下、複合ロックという）も提案されている。以下、それぞれについて簡単に説明する。

【０００３】（１）スピンロック

スピンロックとは、オブジェクトに対してロックを実施＊

```
10 /* ロック */
20 while (compare_and_swap(<o->lock,0,thread_id())==0)
30    yield();
40 /* oへのアクセス */
  . . .
50 /* アンロック */
60 o->lock=0;
```

第２０行及び第３０行でロックを行っている。ロックが獲得できるまでyield()を行う。ここでyield()とは、現在のスレッドの実行を止め、スケジューラに制御を移すことである。通常、スケジューラは、他の実行可能なスレッドから１つを選び走らせるが、いずれまた、スケジュラーは、もとのスレッドを走らせることになり、ロックの獲得に成功するまでwhile文の実行が繰り返される。yieldが存在していると、単にＣＰＵ資源の浪費だけでなく、実装がプラットフォームのスケジューリング方式に依存せざるを得ないため、期待どおりに動作するプログラムを書くことが困難になる。第２０行におけるwhile文の条件であるcompare_and_swapは、オブジェクトｏに用意されたフィールドo->lockの内容と、０とを比較して、その比較結果が真であればスレッドのＩＤ（thread_id()）をそのフィールドに書き込むものである。よって、オブジェクトｏに用意されたフィールドに０が格納されている場合には、ロックしているスレッドが存在しないことを表している。よって、第６０行でアンロックする場合にはo->lockに０を格納する。なお、このフィールドは例えば１ワードであるが、スレッド識別子を格納するのに十分なビット数であればよい。

【０００４】（２）キューロック

キューロックとは、オブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である。キューロックにおいては、スレッドＴがオブジェクトｏのロックに失敗した場合、Ｔは自分自身をｏのキューに入れてサスペンドする。アンロックするコードには、キューが空か否かをチェックするコードが含まれ、空でなければキューから１つスレッドを取り出し、そのスレッドをリジュームする。このようなキューロックは、オペレーティング・システム（ＯＳ）のスケジューリング機構と一体になって実装され、ＯＳのＡＰＩ（Application Programming Interface）として提供されている。例えば、セマフォやＭｕｔｅｘ変数などが代表的なものである。キューロックにおいては、スペースオーバーヘッドはもはや１ワードでは済まず、十数バイトとなるのが普通である。また、ロックやアンロックの関数の内部では、キューという共有資源が操作されるため、何らかのロックが獲得され又は解放されている点にも注意する必要がある。

8

＊するスレッドの識別子を当該オブジェクトに対応して記憶することによりロック状態を管理するロック方式である。スピンロックでは、スレッドＴがオブジェクトｏのロック獲得に失敗した場合、すなわち他のスレッドＳが既にオブジェクトｏをロックしている場合、ロックに成功するまでロックを繰り返す。典型的には、compare_and_swapのようなアトミックなマシン命令を用いて、次のようにロック又はアンロックする。

【表１】

【０００５】（３）複合ロック

マルチ・スレッド対応のプログラムは、マルチ・スレッドで実行されることを考慮して共有資源へのアクセスはロックにより保護するように書かれる。しかし、例えばマルチ・スレッド対応ライブラリがシングル・スレッドのプログラムから使用されるような場合もある。また、マルチ・スレッドで実行されてもロックの競合がほとんど発生しない場合もある。実際、Ｊａｖａ（Sun Microsystems社の商標）のプログラムの実行履歴によると、多くのアプリケーションにおいて、オブジェクトへのアクセスの競合はほとんど発生していないという報告もある。

【０００６】よって、「ロックされていないオブジェクトにロックをかけ、アクセスし、アンロックする」は高頻度に実行されるパスであると考えられる。このパスは、スピンロックでは極めて効率よく実行されるが、キューロックでは時間的にも空間的にも効率が悪い。一方、高頻度ではないとはいえ、競合が実際に発生した場合、スピンロックではＣＰＵ資源が無益に消費されてしまうが、キューロックではそのようなことはない。

【０００７】複合ロックの基本的なアイデアは、スピンロックのような処理が簡単なロック（軽量ロックと呼ぶ）とキューロックのような処理が複雑なロック（重量ロックと呼ぶ）をうまく組み合わせて、上記の高頻度パスを高速に実行しつつ、競合時の効率も維持しようというものである。具体的に言えば、最初に軽量ロックでの

9

ロックを試み、軽量ロックで競合した場合重量ロックに遷移し、それ以降は重量ロックを使用するものである。

【０００８】この複合ロックでは、スピンロックと場合と同様に、オブジェクトにはロック用のフィールドがあり、「スレッド識別子」又は「重量ロック識別子」の値、及び、いずれの値を格納しているかを示すブール値が格納される。

【０００９】ロックの手順は以下のとおりである。

１）アトミックな命令（例えば、compare_and_swap）で軽量ロック獲得を試みる。成功すればオブジェクトへのアクセスを実行する。失敗した場合、すでに重量ロックになっているか、又は軽量ロックのままだが他のスレッドがロックをかけているのかのいずれかであることが分かる。

２）既に重量ロックになっていれば、重量ロックを獲得する。

10

＊３）軽量ロックで競合した場合、軽量ロックを獲得した上で重量ロックへ遷移し、これを獲得する（以下の説明では、inflate関数において実行される。）

【００１０】複合ロックには、３）における「軽量ロックの獲得」でyieldするか否かで２種類の実装がある。これらを詳しく以下に説明する。なお、ロック用のフィールドは１ワードとし、さらに簡単のため「スレッド識別子」又は「重量ロック識別子」は常に０以外の偶数であるとし、ロック用のフィールドの最下位ビットが０ならば「スレッド識別子」、１ならば「重量ロック識別子」が格納される。

【００１１】複合ロックの例１

軽量ロックの獲得において、yieldする複合ロックの場合である。ロック関数は上の手順に従って以下のように書くことができる。

＊　【表２】

```
10: void lock(o) {
20:     if (compare_and_swap(<o->lock, 0, thread_id())
30:         return;
40:     while (! (o->lock < FAT_LOCK)) {
50:         yield();
60:         if (compare_and_swap (<o->lock, 0, thread_id())){
70:             inflate(o);
80:             return;
90:         }
100:    }
110:    fat_lock(o->lock)
120:    return;
130: }
150: void unlock (o){
160:     If (o->lock==thread_id())
170:         o->lock=0;
180:     else
190:         fat_unlock(o->lock);
200: }
220: void inflate(o){
230;     o->lock= alloc_fat_lock() | FAT_LOCK;
240:     fat_lock(o->lock);
250: }
```

【００１２】表２に示された擬似コードは、第１０行から第１３０行までがロック関数、第１５０行から第２００行までがアンロック関数、第２２０行から第２５０行までがロック関数で用いられるinflate関数を示している。ロック関数内では、第２０行で軽量ロックが試みられる。もしロックが獲得されれば、当該オブジェクトへのアクセスを実行する。そして、アンロックする場合には、第１６０行でオブジェクトのロック用フィールドにスレッド識別子が入力されているので、第１７０行においてそのフィールドに０を入力する。このように高頻度パスはスピンロックと同じで高速に実行することができる。一方、第２０行でロックを獲得できない場合には、第４０行でｗｈｉｌｅ文の条件であるロック用フィールドの最下位ビットであるＦＡＴ＿ＬＯＣＫビットとロック用フィールドをビットごとにＡＮＤした結果が０であるか、すなわちＦＡＴ＿ＬＯＣＫビットが０であるか（より詳しく言うと軽量ロックであるか）判断される。もし、この条件が満たされていれば、第６０行にて軽量ロックを獲得するまでyieldする。軽量ロックを獲得した場合には、第２２０行以降のinflate関数を実行する。inflate関数では、ロック用フィールドo->lockに重量ロック識別子及び論理値１であるＦＡＴ＿ＬＯＣＫビ

11

ット入力する（第２３０行）。そして、重量ロックを獲得する（第２４０行）。もし、第４０行で既にＦＡＴ＿ＬＯＣＫビットが１である場合には、直ぐに重量ロックを獲得する（第１１０行）。重量ロックのアンロックは第１９０行にて行われる。なお、重量ロックの獲得及び重量ロックのアンロックは、本発明とはあまり関係ないので説明を省略する。

【００１３】この表２ではロック用フィールドの書き換えは常に軽量ロックを保持するスレッドにより実施される点に注意されたい。これは、アンロックでも同じである。yieldが発生するのは、軽量ロックでの競合時に限＊

12

＊定されている。

【００１４】複合ロックの例２

軽量ロックの獲得において、yieldしない複合ロックの例を示す。軽量ロックが競合した場合にはウエイト（wait）する。軽量ロック解放時には、ウエイトしているスレッドに通知（notify）しなければならない。このウエイト及び通知のためには、条件変数やモニタあるいはセマフォを必要とする。以下の例ではモニタを使用して説明する。

【表３】

```
10: void lock (o) {
20:     if (compare_and_swap (<o->lock, 0, thread_id())
30:        return;
40:     monitor_enter (o);
50:     while (! (o->lock, < FAT_LOCK)){
60:         if (compare_and_swap(<o->lock, 0, thread_id()){
70:            inflate(o);
80:            monitor_exit(o);
90:            return;
100:        } else
110:           monitor_wait(o);
120:    }
130:    monitor_exit(o);
140:    fat_lock(o->lock);
150:    return;
160: }
180: void unlock (o) {
190:     if (o->lock == thread_id()) {
200:        o->lock=0;
210:        monitor_enter(o);
220:        monitor_notify(o);
230:        monitor_exit(o);
240:     } else
250:        fat_unlock(o->lock);
260:  }
280: void inflate (o) {
290:     o->lock = alloc_fat_lock() | FAT_LOCK
300:     fat_lock(o->lock);
310:     monitor_notify_all(o);
320: }
```

【００１５】モニタとは、Ｈｏａｒｅによって考案された同期機構であり、オブジェクトへのアクセスの排他制御（enter及びexit）と所定の条件が成立した場合のスレッドの待機操作（wait）及び待機しているスレッドへの通知操作（notify及びnotify_all）とを可能にする機構である（Hoare, C.A.R. Monitors: An operatingsystem structuring concept. CommunicationS of ACM 17, 10 (Oct. 1974), 549-557　参照）。高だか１つのスレッドがモニタにエンタ（enter）することが許される。スレッドＴがモニタｍにエンタしようとした時、あるスレッドＳが既にエンタしているならば、Ｔは少なくともＳがｍからイグジット（exit）するまで待たされる。このように排他制御がなされる。また、モニタｍにエンタ中のスレッドＴは、ある条件の成立を待つため、モニタｍでウエイト（wait）することができる。具体的には、Ｔは陰にｍよりイグジットしサスペンドする。陰にｍよりイグジットすることにより、別のスレッドがモニタｍにエンタすることができる点に注意されたい。一

13

方、モニタmにエンタ中のスレッドSは、ある条件を成立させた後に、モニタmに通知（notify）することができる。具体的には、モニタmでウエイト中のスレッドのうちのひとつUを起こす（wake up）する。それにより、Uはリジュームし、モニタmに陰にエンタしようとする。ここで、Sがmにエンタ中であるから、Uは少なくともSがmからイグジットするまで待たされる点に注意されたい。また、モニタmでウエイト中のスレッドが存在しない場合には、何も起こらない。notify_allは、ウエイト中のスレッドを全て起こす点を除いて、notifyと同じである。

【００１６】表３において、第１０行乃至第１６０行はロック関数、第１８０行乃至第２６０行はアンロック関数、第２８０行乃至３２０行はinflate関数を示している。ロック関数で複合ロックの例１と異なる点は、第４０行でモニタにエンタする点、軽量ロックで競合した場合にyieldせずにウエイトする点（第１１０行）、重量ロックに遷移した際（第８０行）及び重量ロックに遷移したことが確認された際（第１３０行）にはモニタからイグジットする点である。ここで、第１３０行ではモニタからイグジットし、第１４０行で重量ロックを獲得している点に注意されたい。

【００１７】アンロック関数で複合ロックの例１と異なる点は、第２１０行乃至第２３０行においてモニタにエンタし、モニタで通知をし、モニタをイグジットする処理を実施している点である。これは、yieldをやめてモニタにおけるウエイトにしたためである。inflate関数では、notify_allが追加されている。これもyieldをやめてモニタにおけるウエイトにしたためである。なお、第２９０行は、alloc_fat_lock()で得られる重量ロック識別子と論理値１にセットされたＦＡＴ＿ＬＯＣＫビットをＯＲ操作して、ロック用フィールドに入力する操作を示している。

【００１８】表３を見れば、yieldは消滅しているが、アンロック時にウエイトしているスレッドがいるかもしれないので、通知（notify）という作業が入り、高頻度パスの性能が低下している。また、空間効率的には、モニタ又はモニタと同等な機能が余分に必要になっているが、重量ロックに遷移した後には不要になる。言いかえれば、モニタと重量ロックとは別に用意する必要がある。

【００１９】複合ロックの例３

この例では、複合ロックの例１とは異なり、重量ロックとモニタとを別に用意せず、ＦＡＴ＿ＬＯＣＫビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理をする。例えば、David F.Bacon, Ravi Konuru, Chet Murthy, and Mauricio Serrano. Thin Locks: Featherweight Synchronization for Java. Proceedings of the SIGPLAN '98 Conference on Programming Language Design and Implementation (1998), pp. 258-268を参照のこと。但し、この論文ではyieldが行われている。

14

【００２０】

【発明が解決しようとする課題】本発明の目的は、高頻度パスの処理速度を低下させない、新規な複合ロック方法を提供することである。

【００２１】また、yieldを用いずに、重量ロックとモニタとを別に用意することなく、ＦＡＴ＿ＬＯＣＫビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理できる、オブジェクトのロック方法を提供することも目的である。

【００２２】さらに、上記の複合ロックでは、重量ロックから軽量ロックへの遷移は何等考慮されていない。よって、本発明では、重量ロックから軽量ロックへの遷移を可能にすることも目的である。

【００２３】

【課題を解決するための手段】高頻度パスの処理速度を低下させないようにするため、本発明では少なくとも１ビットの競合ビットを導入する。これはロック用フィールドとは別に用意する。この競合ビットは、軽量ロックで競合が起きた場合にセットされ、軽量ロックから重量ロックへ遷移する際（inflate関数）にクリアされる。これをまとめると、複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又は第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合に、第１のスレッドが保持しているあるオブジェクトへのロックを第２のスレッドが獲得しようとした場合、あるオブジェクトのロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、第１の種類のロックであることを示している場合には、競合ビットを立てるステップとを実行する。

【００２４】このようにすると、第１のスレッドがあるオブジェクトへのロックを解除する際に、ロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、あるオブジェクトのロックを保持しているスレッドが存在しないことを記憶領域に記憶するステップと、ロックの種類を示すビットが第１の種類のロックであることを示している場合には、競合ビットが立っているか判断するステップと、競合ビットが立っていないと判断された場合には、他の処理を実施せずにロック解除処理を終了するステップとを実行できるようになる。すなわち、上で述べた高頻度パスの性能低下が競合ビットが立っているか否かのチェックだけに限定される。

【００２５】一方、競合ビットが立っていると判断された場合には、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機

15

しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、待機しているスレッドへの通知操作を実行するステップと、第１のスレッドが排他制御状態から脱出するステップとを実施する。

【００２６】なお、第１の種類のロックとは、オブジェクトに対してロックを実施するスレッドの識別子を当該オブジェクトに対応して記憶することによりロック状態を管理するロック方式であり、第２の種類のロックとは、オブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である。

【００２７】また、yieldを用いずに、重量ロックとモニタとを別に用意することなく、ＦＡＴ＿ＬＯＣＫビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理できるようにするため、複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合には、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、あるオブジェクトのロックの種類を示すビットが第１の種類のロックであることを示しているか判断するステップと、あるオブジェクトのロックの種類を示すビットが第１の種類のロックであることを示している場合、第１の種類のロックを第１のスレッドが獲得できるか判断するステップと、第１の種類のロックを第１のスレッドが獲得できる場合には、あるオブジェクトに対応して設けられた記憶領域に第２の種類のロックを示すビット及び第２の種類のロックの識別子を記憶するステップとを実行し、第１のスレッドはあるオブジェクトに対して必要な処理を終了した後に排他的状態を脱出する。

【００２８】第１の種類のロックを第１のスレッドが獲得できない場合には、モニタの待機状態に移行するステップをさらに実行する。あるオブジェクトのロックのロックの種類を示すビットが第１の種類のロックであることを示していない場合、第１のスレッドは第２の種類のロックを獲得したとして排他的状態を脱出することなく処理を実施するステップとをさらに実行する。

【００２９】また、上記目的のため、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第１のスレッドが移行するステップと、あるオブジェクトのロックの種類を示すビットが第２の種類のロックであることを示しているか判断するステップと、あるオブジェクトのロックの種類を示すビットが第２の種類のロックであることを示している場合、第１のスレッドは前記第２の種類のロックを獲得したとして排他的状態を脱出することなく処理を実施するステップとを実行する。既に、重量ロックに遷移している場合には、モニタにエンタすることにより重量ロックの獲得として処理を実施できる。

16

【００３０】重量ロックから軽量ロックへの遷移を可能にするために、複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第１の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第２の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、ロックを解除するには、第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックであるか判断するステップと、第１のスレッドが獲得しているあるオブジェクトのロックが第２の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが存在しているか判断するステップと、他のスレッドが存在していない場合、所定の条件を満たしているか判断するステップと、所定の条件を満たしている場合に、ロックを保持しているスレッドが存在しないことを記憶領域に記憶するステップとを実行する。

【００３１】以上述べた処理のフローは、専用の装置として実施することも、また、コンピュータのプログラムとして実施することも可能である。さらに、このコンピュータのプログラムは、ＣＤ－ＲＯＭやフロッピー・ディスク、ＭＯ（Magneto-optic）ディスクなどの記憶媒体、又はハードディスクなどの記憶装置に記憶される。

【００３２】

【発明の実施の形態】本発明の処理が実施されるコンピュータの例を図１に示す。コンピュータ１０００は、ＣＰＵ（１又は複数）及びメインメモリを含むハードウエア１００と、ＯＳ（Operating System）２００、アプリケーション・プログラム３００を含む。ＯＳ２００は、アプリケーション・プログラム３００として動作する複数のスレッドを可能にする能力を有する。また、ＯＳ２００はキューロックに必要な機能も提供する。また、アプリケーション・プログラム３００は、モニタ機能、本発明のロック及びアンロック機能を含む。さらに、Ｊａｖａ言語の場合には、Ｊａｖａ　ＶＭ（Virtual Machine）３１０をＯＳ２００上に設け、さらにその上でアプレット又はアプリケーション３２０を実行する場合もある。アプレット又はアプリケーション３２０もマルチ・スレッドで実行され得る。Ｊａｖａ言語においては、Ｊ

17

ａｖａ　ＶＭ３１０に、モニタ機能、本発明のロック及びアンロック機能が組み込まれる。なお、Ｊａｖａ　ＶＭ（３１０）はＯＳ２００の一部として組み込まれる場合もある。また、コンピュータ１０００は補助記憶装置を有しない、いわゆるネットワークコンピュータ等でもよい。

【００３３】本発明の処理を説明前に、高頻度パスの処理速度を低下させないための競合ビットを新たに導入する。図２に示したように、あるオブジェクトをロックしているスレッドが存在しない場合（（１）の場合）には、ロック用フィールド及び競合ビット共に０が格納される。その後、あるスレッドがそのオブジェクトをロック（軽量ロック）すると、そのスレッドの識別子がロック用フィールドに格納される（（２）の場合）。もし、

18

このスレッド識別子のスレッドがロックを解放するまでに他のスレッドがロックを試みなければ（１）に戻る。ロックを解放するまでに他のスレッドがロックを試みると、軽量ロックにおける競合が発生したので、この競合を記録するため競合ビットを立てる（（３）の場合）。その後、重量ロックに移行した際には、競合ビットはクリアされる（（４）の場合）。可能であれば、（４）は（１）に移行する。なお、ロック用フィールドの最下位に軽量ロックと重量ロックのモードを表すビット（ＦＡＴ＿ＬＯＣＫビット）設けるようにしたが、最上位に設けるようにしても良い。

【００３４】上のような競合ビット及びロック用フィールドを用いた本発明の処理を以下に示す。

【表４】

```
10: void lock (Object* o){
20:     /* 軽量ロック */
30:     if (compare_and_swap (<o->lock, 0, thread_id()))
40:         return;
50:     /* 重量ロック及びモード遷移パス */
60:     MonitorId mon=obtain_monitor(o);
70:     monitor_enter(mon);
80:     /* モード遷移ループ */
90:     while (o->lock < FAT_LOCK) ==0) {
100:      set_flc_bit(o);
110:      if (compare_and_swap (<o->lock, 0, thread_id()))
120:         inflate(o, mon);
130:      else
140:         monitor_wait (mon);
150:    }
160:
170: }
180:
190: void unlock (Object* o) {
200:     /* 軽量ロックパス */
210:     if ((o->lock < FAT_LOCK) ==0)
220:         o->lock=0;
230:         if (test_flc_bit(o)) { /* 本発明のオーバーヘッド */
240:             MonitorId mon=obtain_monitor(o);
250:             monitor_enter(mon);
260:             if (test_flc_bit(o))
270:                 monitor_notify(mon);
280:             monitor_exit(mon);
290:         }
300:         return;
310:     }
320:     /* 重量ロックパス */
330:     x=o->lock
340:     if (oについてモニタでウエイトしているスレッドがない)
350:         if (所定の条件が満たされた)
360:             o->lock=0; /* 重量ロックから軽量ロックへの遷移 */
```

19　　20

```
370:     monitor_exit( x < ~FAT_LOCK );
380: }
390:
400:
410: void inflate (Object* o, MonitorId mon) {
430:     monitor_notify_all (mon);
440:     o->lock= (Word) mon | FAT_LOCK;
450: }
460:
470:
480: MonitorId obtain_monitor(Object* o){
490:     Word word=o->lock;
500:     MonitorID mon;
510:     if (word < FAT_LOCK)
520:          mon = word < ~FAT_LOCK;
530:     else
540:          mon = lookup_monitor(o);
550:     return mon;
560: }
```

【００３５】本発明で導入された競合ビットは表４ではｆｌｃ＿ｂｉｔとして示されている。では、表４の内容を詳細に説明する。表４は大きく分けて４つの部分からなる。ロック関数の部分（第１０行乃至第１７０行）、アンロック関数の部分（第１９０行乃至第３８０行）、軽量ロックから重量ロックへの遷移であるinflate関数の部分（第４１０行乃至第４５０行）、及びモニタの識別子を取得するobtain_monitor関数の部分（第４８０行乃至第５６０行）である。

【００３６】（１）ロック関数

第１０行から始まったオブジェクトｏに対するロック関数の処理では、まず軽量ロックの取得を試みる（第３０行）。この軽量ロックの取得には、例えばcompare_and_swapのようなアトミックな命令を用いる。この命令では、第１の引き数と第２の引き数が同じ値の場合、第３の引き数を格納するものである。ここでは、オブジェクトｏのロック用フィールドであるo->lockが０に等しい場合には、thread_id()によりスレッド識別子を取得して、ロック用フィールドo->lockに格納する。図２の（１）から（２）への遷移を実施したのである。そして、必要な処理を実施するため、リターンする（第４０行）。もし、オブジェクトｏのロック用フィールドであるo->lockが０に等しくない場合には、軽量ロックの取得は失敗し、第６０行に移行する。ここまでの処理は従来技術と同じである。

【００３７】次に、モニタ識別子を取得するobtain_monitor(o)関数の値をmonという変数に代入し（第６０行）、スレッドはそのモニタの排他制御状態に移行しようとする。すなわちモニタ（monitor）にエンタ（enter）しようとする（第７０行）。もし、排他制御状態に移行することができれば、以下の処理を実施し、もしできなかった場合には、できるまでこの段階で待つ。次に、while文の条件を判断する。すなわち、ロック用フィールドo->lockとＦＡＴ＿ＬＯＣＫビットのビットごとのＡＮＤを実施し、ＦＡＴ＿ＬＯＣＫビットが立っているか判断する（第９０行）。ここでは、現在重量ロックに移行しているのか、軽量ロック中なのかを判断している。もし、ＦＡＴ＿ＬＯＣＫビットが立っていなければ（軽量ロック中）、この計算の結果は０となるから、while文以下の処理を実施する。一方、ＦＡＴ＿ＬＯＣＫビットが立っている場合（重量ロック中）、while文以下の処理を実施せずに、モニタにエンタした状態のままになる。このようにＦＡＴ＿ＬＯＣＫビットが立っている場合に、モニタにエンタできた場合には、本発明では重量ロックを取得できたということを意味しており、このモニタからイグジット（exit）することなく（すなわち排他制御状態を脱出することなく）、このスレッドはオブジェクトに対する処理を実施する。

【００３８】では、第９０行でＦＡＴ＿ＬＯＣＫビットが立っていないと判断された場合には、軽量ロックの競合が発生していることを意味するので、ｆｌｃ＿ｂｉｔをセットする（第１００行、set_flc_bit(o)）。ここで、図２の（２）から（３）への遷移を実施したのである。そして、もう一度軽量ロックを取得できるか判断する（第１１０行）。もし、軽量ロックを取得できる場合には軽量ロックから重量ロックへの遷移のためのinflate関数の処理を実施する（第１２０行）。一方、軽量ロックが取得できない場合には、モニタの待機状態（wait）に移行する（第１４０行）。モニタの待機状態は、先にモニタの説明の部分で述べたが、モニタから脱出してサスペンドするものである。このように、軽量ロックで競合が生じると、競合ビットであるｆｌｃ＿ｂｉｔが

21

セットされ、軽量ロックを取得できない場合には、モニタの待機状態に移行する。この待機状態に入ると、後にinflate関数の処理又はアンロックする際に通知（notify又はnotify_all）を受けることになる。

【００３９】（２）inflate関数

では、第４１０行乃至第４５０行のinflate関数の処理を説明する。ここではまず、競合ビットがクリアされる（第４２０行、clear_flc_bit）。そして、モニタの通知操作（monitor_notify_all）を実施する（第４３０行）。ここでは、待機状態の全てのスレッドに起きる（wake up）よう通知する。そして、ロック用フィールドo->lockに、モニタの識別子を格納した変数ｍｏｎとセットされたＦＡＴ＿ＬＯＣＫビットをビットごとにＯＲした結果を格納する（第４４０行、mon | FAT_LOCK）。すなわち、図２の（３）から（４）の状態に遷移させたのである。これで軽量ロックから重量ロックへの遷移は完了する。なお、第１２０行の処理が終了すると、再度while文の条件をチェックすることになるが、既にＦＡＴ＿ＬＯＣＫビットが立っているので、この場合にはwhile文から脱出して、モニタにエンタしたままとなる。すなわち、while文の中の処理を実行しない。

【００４０】通知を受けた全てのスレッドは第１４０行において陰にモニタにエンタしようとするが、モニタにエンタする前に待機することになる。これは、通知を行ったスレッドはアンロック処理を実施するまでモニタからイグジットしていないからである。

【００４１】（３）アンロック関数

では、次に第１９０行乃至第３８０行のアンロック関数の処理について説明する。このアンロック関数は軽量ロックのアンロックと、重量ロックのアンロックを取扱う。重量ロックにおけるアンロックは、図２の（４）から（１）への遷移を取扱うものである。

【００４２】（３－１）軽量ロックのアンロック

軽量ロックのアンロックでは、まず、ロック用フィールドo->lockとＦＡＴ＿ＬＯＣＫビットのビットごとのＡＮＤを計算し、その値が０であるか判断する（第２１０行）。これは、第９０行のwhile文の条件と同じであって、軽量ロック中であるかどうか判断するものである。もし、軽量ロック中である場合には、o->lockに０を格納する（第２２０行）。これにより、ロックを保持しているスレッドが存在しないことが記録される。そして、競合ビットが立っているか判断する（第２３０行、test_flc_bit）。もし、軽量ロックで競合が生じていなくとも、第２３０行のみは実施しなければならない。よって、本発明における高頻度パスの唯一のオーバーヘッドがこの第２３０行である。競合ビットが立っていない場合には、他の処理を実施せずにアンロック処理を終了する（第３００行）。

【００４３】もし、競合ビットが立っている場合には、第６０行及び第７０行と同じように、変数ｍｏｎにモニタの識別子を格納し（第２４０行）、当該モニタ識別子のモニタにエンタしようとする（第２５０行）。すなわち、そのスレッドはモニタの排他制御状態に入ろうとする。もしモニタにエンタできた場合には、もう一度、競合ビットが立っていることを確認し（第２６０行）、もし立っていれば、モニタにおいて待機状態のスレッドの１つに起動を通知する（第２７０行、monitor_notify(mon)）。なお、モニタにエンタできない場合には、モニタにエンタできるまで待機する。そして通知を行ったスレッドは、モニタの排他制御状態から脱出する（第２８０行、monitor_exit(mon)）。

22

【００４４】第２７０行で通知を受けたスレッドは、第１４０行で陰にモニタにエンタする。そして第８０行に戻りその処理を実施する。通常、第２７０行で通知を受けたスレッドは、通知を行ったスレッドがモニタの排他制御状態を脱出した後にモニタの排他制御状態に入り、競合ビットを立てた後に、軽量ロックを取得し、inflate関数の処理を実施することにより重量ロックに遷移する。

【００４５】（３－２）重量ロックのアンロック

もし、第２１０行でＦＡＴ＿ＬＯＣＫビットが立っていることが分かった場合には、第３３０行に処理は移行する。第３３０行では、ロック用フィールドの内容を変数ｘに格納する。そして、モニタにおける待機状態（wait）にあるスレッドが他に存在しないかを判断する（第３４０行）。もし、存在しない場合には、所定の条件を満たしているか判断する（第３５０行）。所定の条件には、重量ロックから脱出しない方が良いような条件があればそのような条件を設定する。但し、本ステップは実行しなくてもよい。もし、所定の条件を満たしている場合には、ロック用フィールドo->lockを０にする（第３６０行）。すなわち、ロックを保持しているスレッドが存在しないことをロック用フィールドに格納する。そして、変数ｘのＦＡＴ＿ＬＯＣＫビット以外の部分に格納されたモニタ識別子のモニタからイグジットする（第３７０行）。x < ~FAT_LOCK は、FAT_LOCKビットを反転させたものとxとのビットごとのＡＮＤである。これにより、モニタにエンタしようとして待機していたスレッドが、モニタにエンタできるようになる。

【００４６】（４）モニタ識別子を取得するobtain_monitor関数

この関数では、まず、wordという変数にロック用フィールドの内容を格納する（第４９０行）。そして、モニタの識別子を格納する変数ｍｏｎを用意し（第５００行）、ＦＡＴ＿ＬＯＣＫビットが立っているか判断する（第５１０行、word< FAT_LOCK）。もし、ＦＡＴ＿ＬＯＣＫビットが立っているようであれば、変数ｍｏｎにｗｏｒｄのＦＡＴ＿ＬＯＣＫビット以外の部分を格納する（第５２０行、word < ~FAT_LOCK）。一方、ＦＡＴ＿ＬＯＣＫビットが立っていない場合には、関数ｌｏｏｋｕ

23

ｐ＿ｍｏｎｉｔｏｒ（ｏ）を実行する（第５３０行）。この関数は表４で説明は省略しているが、オブジェクトとモニタの関係を記録したハッシュ・テーブルを有していることを前提とし、基本的にはこのテーブルをオブジェクトｏについて検索して、モニタの識別子を取得する。もし、必要があれば、モニタを生成し、そのモニタの識別子をハッシュ・テーブルに格納した後にモニタ識別子を返す。いずれにしても、変数ｍｏｎに格納されたモニタの識別子を返す。

【００４７】従来技術の欄で説明した従来技術と表４を比較すると、競合ビットを導入した他に、第１５０行乃至第１７０行の間に何等の処理が存在していない点、及び第３２０行乃至第３７０行の重量ロックから軽量ロックへの遷移が存在している点、が大きく異なる。競合ビットを導入したことにより第２３０行のチェックが必要になったが、競合ビットを導入しなければ、従来技術のような、より大きなペナルティを受ける。また、ＦＡＴ＿ＬＯＣＫビットが立っており且つモニタの排他制御状態に移行することができた場合には重量ロックを獲得しているということにしたため、モニタの他に重量ロックの機構を用意する必要がなくなり、且つモニタの排他的状態からの脱出及び重量ロックの獲得といった処理をなくし、それにより処理を高速化することもできるようになった。また、重量ロックから軽量ロックへの遷移（図２の（４）から（１））を設けたことにより、低負荷な高頻度パス（図２の（１）と（２）の間の遷移）を実行できるような状態に戻ることができた。

【００４８】以下に、競合ビットを表４内の第１００行でセットし、第２３０行でチェックすることで何等の問題が生じないということについて述べておく。最初に、「競合ビットは、inflate関数でのみクリアされる」ということを確認しておく。

【００４９】そして、スレッドＴがウエイト（wait）したとする。スレッドＴが必ず通知（notify）を受けることを、次の２つの場合に分けて説明する。

（１）その後inflate関数が実行される場合。

inflate関数が実行されると、第４３０行目でnotify_allが実行される。すなわち、Ｔはnotifyを受ける。

（２）inflate関数が実行されない場合。

Ｔがウエイト（wait）したのは、第１１０行目における軽量ロック獲得に失敗したからである。第１１０行目の失敗の時点を考えると、この時点で別なスレッドＳが軽量ロックを保持している、すなわち、Ｓはアンロック関数の第２２０行目の実行に到達していない。また、Ｔがウエイト（wait）前にセットした競合ビットは、inflate関数が実行されない場合を考えているので、上で確認

24

した事項により、セットされたままである。Ｓはいずれアンロック関数の第２２０行目に到達し、次の競合ビットのチェックを実行するが、このチェックは必ず成功する。すなわち、ＴはＳにより通知（notify）される。

【００５０】また、図２における（４）から（１）の遷移を導入した。これは、１）軽量ロックを獲得するためには第３０行のcompare_and_swapを成功しなければならないが、他のスレッドが重量ロックを獲得している限り、第３０行のcompare_and_swapは成功しないので、重量ロックを他のスレッドが獲得している時には軽量ロックを獲得することは不可能であることが保証され、２）重量ロックを獲得するためには、モニタにエンタしてwhile文の条件が満たされない必要があるが、他のスレッドが軽量ロックを保持している限り、必ずwhile文の条件が満たされてしまうので、軽量ロックを他のスレッドが獲得している時にはモニタにエンタできても重量ロックを獲得することは不可能であることが保証されるので、安全な処理である。

【００５１】

【効果】高頻度パスの処理速度を低下させない、新規な複合ロック方法を提供することができた。

【００５２】また、yieldを用いずに、重量ロックとモニタとを別に用意することなく、ＦＡＴ＿ＬＯＣＫビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理できる、オブジェクトのロック方法を提供することもできた。

【００５３】さらに、上記の複合ロックでは、重量ロックから軽量ロックへの遷移は何等考慮されていない。よって、本発明では、重量ロックから軽量ロックへの遷移を可能にすることもできた。

【図面の簡単な説明】

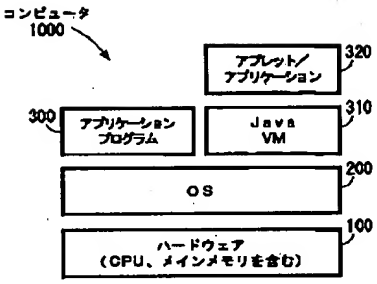
【図１】本発明の処理が実施されるコンピュータの一例を示す図である。

【図２】モードの遷移、並びに各モードにおけるロック用フィールド（ＦＡＴ＿ＬＯＣＫビットを含む）及び競合ビットの状態を説明するための図である。なお、（１）はロックなし、（２）は軽量ロックで競合なし、（３）は軽量ロックで競合あり、（４）は重量ロックの状態を示す。

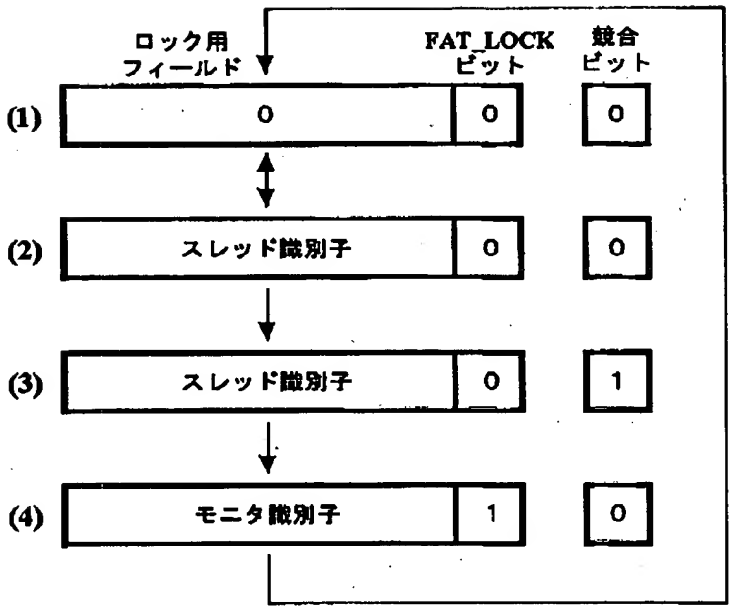
【符号の説明】

１０００　コンピュータ
１００　ハードウエア
２００　ＯＳ
３００　アプリケーション・プログラム
３１０　Ｊａｖａ　ＶＭ
３２０　Ｊａｖａ　アプレット／アプリケーション

【図１】



【図２】



フロントページの続き

(72)発明者　小野寺　民也
神奈川県大和市下鶴間1623番地14　日本アイ・ビー・エム株式会社　東京基礎研究所内

(72)発明者　河内谷　清久仁
神奈川県大和市下鶴間1623番地14　日本アイ・ビー・エム株式会社　東京基礎研究所内

Ｆターム(参考)　5B098 GA05 GD16